VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Rozšíření projektu Other Kosmos o správu uživatelů a správu NPC

Players and NPC management for Project Other Kosmos

Denis Lokaj

Bakalářská práce

Vedoucí práce: Ing. David Ježek, Ph.D.

Ostrava, 2021

Abstrakt

Cílem práce je rozšířit projekt Other Kosmos o možnost registrace uživatelů a správu samostatně se chovajících postav (NPC) a jejich začlenění do simulovaného světa. Nedílnou součásti řešení je analýza současného stavu, návrh nové funkcionality a její implementace.

Klíčová slova

Java, LibGDX, EclipseLink, JavaFX, vývoj her, hráčská postava, nehratelná postava, uživatelské rozhraní, databáze

Abstract

The aim of the work is to expand the project Other Cosmos with the possibility of user registration and management of self-behaving characters (NPC) and their integration into the simulated world. An integral part of the solution is the analysis of the current state, the design of new functionality and its implementation.

Keywords

Java, LibGDX, EclipseLink, JavaFX, game development, player character, non-playable character, graphical user interface, database

Poděkování

Rád bych na tomto místě poděkoval Ing. Davidovi Ježkovi, Ph.D a rodině, která mi s prací pomohla, protože bez nich by tato práce nevznikla.

Obsah

Seznam použitých zkratek a symbolů

RPG	–	Role-playing games
NPC	–	Non-playable character
JVM	–	Java virtual machine
GC	–	Garbage Collector
ORM	–	Object–relational mapping
JPA	–	Java Persistence AP
GUI	–	Graphical user interface
PC	–	Personal computer

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tato práce je rozšířením už započatého herního projektu Other Kosmos, na této hře pracovalo už spoustu studentů, hra je inspirována známou hrou Minecraft, kterou vytvořilo Švédské herní studio Mojang. Stejně jako Minecraft je pro vývoj Other Kosmos použit populární objektově orientovaný jazyk Java s rozšířením herního enginu LibGDX. Herní svět nabývá tvar kostek, tzv. „Voxelů“, což může místy působit jako jakési lego a v hráči vyvolat touhu s těmahle kostkama manipulovat.

Mým úkolem bylo využít databázový engine H2 ve spojení s technologií EclipseLink a poskytnout řešení při ukládání uživatelských účtů a jejich jednotlivých herních postav a začlenění jejich postupu do hry. Dále bylo potřeba rozšířit implementaci NPC, na které pracoval můj kolega ve své diplomové práci minulý rok. Práce obsahuje i jednoduché prostředí pro správu vytvořené v JavaFX.

Řešení, které jsem navrhl je postupem času lehce rozšiřitelné, při nabytí nových vlastností jednotlivých postav bude snadné jejich informace ukládat už do předem předpřipravených tabulek v databázi.

Kapitola 2

Herní postavy a počítačem kontrolované postavy

2.1 Hry na hrdiny

Herní postavy se jako první vyskytly ve stolních hrách na hrdiny, anglicky zvané „RPG“ tzv. Role-playing games. V těchto hrách hráči zaujímají pozice fiktivních postav, které musí dodržovat určitá pravidla hry a tyto pravidla hry jsou kontrolována tzv. „pánem hry“, anglicky game master. Jeho úlohou je sehrát určitou inteligenci ve hře, tudíž vytvořit vazby mezi jednotlivými postavami a kontrolovat nehráčské aspekty hry. Většinou je to také hlavní „soudce“ při sporech mezi jednotlivými hráči. Občas se může role pána hry rozdělit mezi všechny hráče a hráči poté kolektivně rozhodují o sporných situacích např. hlasováním. Někdy se tato role může předávat po uplynutí určitého časového úseku, tudíž každý hráč si zkusí jaké to je být pánem hry. V důsledku toho dojde k lepšímu osvojení si této role a nabrání zkušeností do dalších her.

Anglický termín NPC, značí postavy nekontrolované hráčem, termín byl poprvé užit v souvislosti s Role-playing deskovými hrami, kdy pán hry tyto postavy kontroloval. [1]

2.2 Dračí doupě

Dračí doupě je česká fantasy RPG hra, vydána nakladatelstvím Altar v roce 1990, přejímá nápady z komerčně úspěšné zahraniční stolní hry Dungeons and Dragons.

Zde se pán hry nazývá Pán Jeskyně, před zahájením hry je jeho úkolem připravit úvodní příběh a mapu. Každý z hráčů má vlastní postavu ve fantasy světě, např. elfí čaroděj. Hráči se poté vydávají na společnou cestu herním světem a jsou jim zadávány různé úkoly, jejich cílem je tyto úkoly plnit a získat za ně odměny.

Herní mechaniky jako boj, kouzla a zvláštní schopnosti se řeší pomocí číselných tabulek uvedených v pravidlech. Mechanika, která má hru oživit, je prvek náhody, zde je zastoupen hodem

kostkou. Kostek je povícero druhů, používají se jak klasické šestistěnné, tak i desetistěnné nebo ve vyjímečných případech i osmi- či čtyřstěnné. Typicky se nejčastěji hraje na čtverečkovaném papíře.

Pravidla Dračího Doupěte jsou podrobně popsány v šesti různých knihách, některé jsou určeny pro Pána Jeskyně a zbylé pro hráče, dohromady tvoří kolem pětisetpadesáti stran. [2]

Obrázek 2.1: RPG hrací kostky [3]

2.3 Nehráčské postavy

Nehráčské postavy, neboli anglicky NPC jsou postavy, které jsou součástí herního prostředí a slouží k interakci s hráčem kontrolovanými postavami. Tento termín byl rovněž prvně použit v souvislosti s Role-playing deskovými hrami.

V konzolových nebo počítačových hrách slouží termín k popsání entity, která není pod přímou kontrolou hráče. NPC jsou většinou označeny ty entity, které jsou přátelsky případně neutrálně vyhraněna k hráči. Termíny jako „mobky“, „creepeři“ označují ty postavy, které jsou vůči hráčům nepřátelské. [4]

2.3.1 Historie umělé inteligence ve hrách

Chování NPC je ve videohrách naprogramováno, tvoří takzvanou „Umělou inteligenci“ anglicky nazývanou Artificial Intelligence neboli AI. Cílem je co nejpřesněji napodobit chování entit z našeho reálného života.

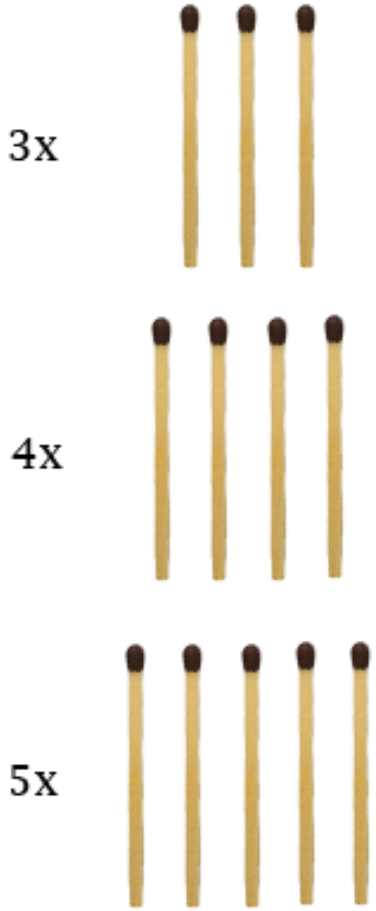
Jako nejstarší příklad využití umělé inteligence ve hrách můžeme uvést hru „Nim“ (Obr. 2.2), která byla vytvořena v roce 1951 firmou Ferranti podle teorie hry Charlese L. Boutona. Obecně se má za to, že hra má svůj původ v Číně. Jde o kombinatorickou hru pro 2 hráče. Princip spočívá v odebírání zápalek ze skupinky. V každém tahu je hráči umožněno odebrat několik zápalek, minimálně jednu a maximum je omezeno pravidly konkrétní hry. Hráč, který odebere poslední zápalku vyhrává, případně prohrává, záleží na konkrétních pravidlech varianty. [5]

2.3.2 Moderní umělá inteligence ve hrách

Videohry, které měly k dispozici tzv. „single-player“ mód s nepřáteli se začaly objevovat v 70. letech. Jako jeden z nejznámějších příkladů můžeme uvést Qwak od známé videoherní společnosti Atari, kde je hlavní pointou hry střílet kachny, které svou umělou inteligencí simulují let ve vzduchu a případný dopad na zem při sestřelení. Od té doby byl podobný koncept použit několikrát a iterace této hry nalezneme i v dnešní době na různých mobilních zařízeních nebo virtuální realitě.

NPC ještě nejčastěji využívají tzv. rozhodovací stromy, což jsou ručně napsané pravidla chování, které mají za následek určité nedostatky v logice, jako je opakovatelnost a neočekávané chování v situacích, které vývojáři nezamýšleli.

Hledání cesty anglicky pathfinding je další forma umělé inteligence, nejčastěji používaná v souvislosti s realtime strategiemi. Jedná se o metodu, kdy je potřeba NPC dostat z jednoho bodu do druhého a přitom brát v úvahu terén a překážky. [6] [7]



Obrázek 2.2: hra Nim [5]

2.4 Minecraft

Minecraft je sandboxová videohra vytvořena herním studiem Mojang v programovacím jazyku Java. Později odkoupena Microsoftem.

V minecraftu hráči prozkoumávají procedurálně generovaný svět z bloků s nekonečným terénem a můžou zde interagovat s různým materiálem, pomocí kterého lze v kombinaci s jiným vytvářet různé předměty. Hra nabízí různé módy, kdy kreativní je o tom, že můžeme stavět a vytvářet co chceme s nekonečným stavebním materiálem, bez toho aniž bychom byli omezování nepřátelskými creepery. Mód „přežití“ je pravý opak módu kreativního, kdy musíme dávat pozor na své životy, hlad a zásoby. Cílem hry je zabít Ender draka, ovšem hra i nadále po splnění tohoto úkolu pokračuje dál. [8]

Obrázek 2.3: Ukázka ze hry Minecraft

Kapitola 3

Použité technologie

3.1 Java

Celý projekt Other Kosmos je napsaný v jazyce Java a o specifických vlastnostech tohoto jazyka se chci bavit v této kapitole.

Tento programovací jazyk byl vyvinut firmou Sun Microsystems a v dnešní době se jedná o jeden z nejvíce používaných objektově orientovaných jazyků. Syntaxe jazyka je podobná jazyku C a C++, ovšem zde nejsou přítomny např. ukazatele. Java nabízí výhodu toho, že je multiplatformní, tudíž pokud je na zařízení dostupná „Java Virtual Machine“ neboli JVM, je možné na tomto zařízení spustit programy vyvíjené v tomto jazyce, to platí nejen pro desktopové zařízení(X86), ale i mobilní(ARM).

JVM je sada virtuálních nástrojů, které byly vyvinuty z důvodu, aby na nich běžely Java aplikace, ovšem v dnešní době je možno na JVM spustit i aplikace vyvinuté v jiných jazycích, jako je např. Python, Ruby a další. Taktéž nabízí automatickou správu paměti. To znamená při vývoji softwaru jistou výhodu, jelikož se programátoři nemusí starat o pamět. Aplikace nevyužívající garbage collector často padaly z důvodu tzv. memory leak a bylo časové i finančně náročné u velkých projektů zjistit, kde se v kódu nachází chyba. Problém to byl zejména u programů, které musely fungovat téměř nepřetržitě.

Funkčnost technologie Garbage Collector je ve vyhledávání nepoužívaných objektů v paměti haldy JVM. Java obsahuje čtyři typy implementace GC.

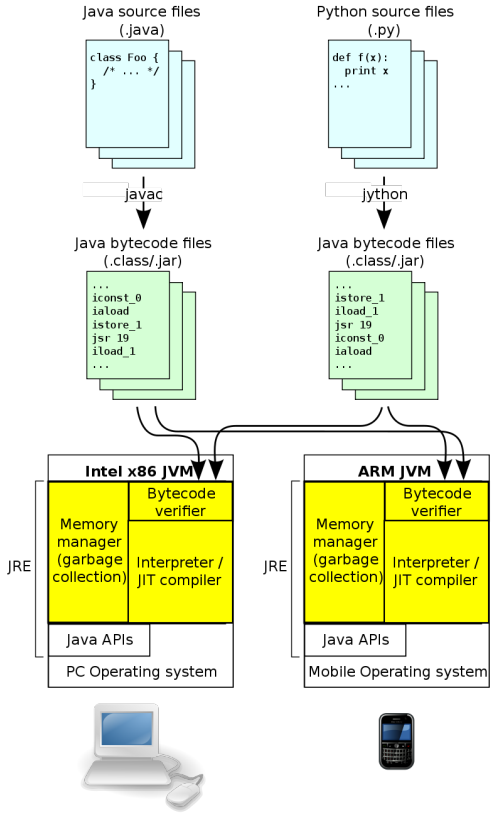
- Serial Garbage Collector - Jedná se o nejjednodušší implementaci GC. Pracuje s jedním vláknem, v důsledku toho, tento typ pozastaví ostatní vlákna když se spustí. Tento typ není vhodný pro aplikace vyžadující výkon v reálném čase např. vojenské senzory.

- Parallel Garbage Collector - Jde o výchozí GC v JVM. Funguje na stejném principu jako Serial GC, ovšem využívá více vláken.

- CMS Garbage Collector - Je vhodný pro aplikace vyžadující krátké pauzy. Výkonostně je na tom hůř, zato nedochází k zastavení aplikace, pouze k jejímu zpomalení. V novějších verzích Java neposkytuje podporu pro tento GC a je nahrazen G1 GC.

- G1 Garbage Collector - Poskytuje lepší výkon než předchozí typ GC. Je vhodný pro víceprocesorové systémy s velkou pamětí.

[9] [10] [11] [12] [13]



Obrázek 3.1: Java Virtual Machine [14]

3.2 JavaFX

JavaFX je softwarová platforma, která slouží k vývoji Graphical user interface, zkráceně GUI. Byla vyvinuta jako náhrada za už zastaralý Swing, který neposkytoval možnosti vývoje moderního uživatelského rozhraní. JavaFX nabízí nástroje pro vývoj internetových nebo desktopových aplikací. [15]

3.3 LibGDX

LibGDX je open source a volně dostupný framework pro tvorbu herních aplikací. Je napsán v programovacím jazyce Java s některými výkonostně závislými komponenty napsanými v C nebo C++. Podporuje vývoj mobilních, ale i počítačových her, je multiplatformní co se operačních systémů týče. Na jeho základě stojí celý vývoj Other Kosmos. [16]

3.4 Scene Builder

Scene Builder je vizuální nástroj, který umožňuje vytváření grafického uživatelského rozhraní v JavaFX pomocí technologie drag and drop, což znamená, že uživatelé jen přetahují jednotlivé komponenty na pracovní plochu a dále nastavují jejich vlastnosti a aplikují styly, na pozadí se generuje kód, který je poté uložen do souboru FXML. Jednotlivým komponentám lze přiřazovat jména proměnných a následně s nima pracovat dále. [17]

3.5 Git

Git je open-source software pro sledování změn souborů, je používán pro koordinaci práce mezi programátory, který umožnuje paralelní vývoj a následné spojení různých části kódu ve výsledný funkční produkt. Tento software byl vyvinut Linusem Torvaldsem, autorem Linuxu právě pro zefektivnění vývoje Linuxu. Do té doby byl využívan systém správy zdrojového kódu BitKeeper, kdy vývojáři Linuxu měli možnost tento systém používat bezplatně. Ovšem po incidentu s Andrewem Tridgellem, který pomocí reverzního inženýrství vytvořil vlastní obdobný systém, byla licence o užití s vývojáři Linuxu ukončena. [18]

Mezi základní vlastnosti verzovacího systému Git patří:

- Snímkování - Git uvažuje o svých datech jako o sérii snímku, kdy identický soubor už znovu neukládá, pouze na něho odkazuje.
- Lokální - Git pracuje s lokálními soubory, tudíž práce není ovlivněna latencí. Veškerá historie projektu je uložena v počítači, lze tedy pracovat offline a po provedení práce pouze odeslat aktualizované soubory na vzdálený server.
- Integrita - Není možné změnit obsah souboru bez toho, aniž by o tom Git věděl, používá otisky podle kterých ukládá jeho obsah.
- Přidávání dat - Git téměř všechny data pouze přidává, nemaže, lze tedy jednoduše data obnovit a neztratit svou práci.
- Tři stavy

– Změněno (modified) znamená, že v souboru byly provedeny změny, ovšem nebyly ještě zapsány do databáze.

– Připraveno k zapsání (staged) znamená, že byl změněný soubor v aktuální verzi určen k zapsání do dalšího snímku (commit snapshot).

– Zapsáno (commit) znamená, že jsou data uložena v lokální databázi.

3.6 EclipseLink

Jedná se o technologii umožňující automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem, zkráceně ORM. EclipseLink je implementovaný podle standartu programovacího jazyka Java nazývaný Java Persistence API, zkráceně JPA. [19]

3.7 JBCrypt

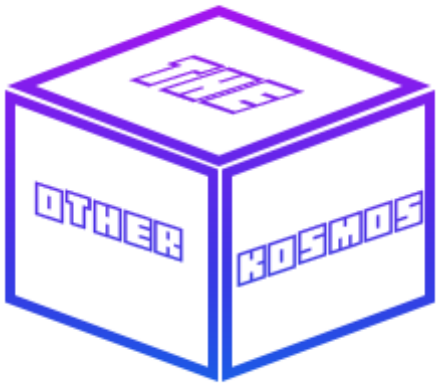
BCrypt je hašovací funkce vyvinutá Nielsem Provosem a Davidem Mazièresem, jde o vylepšenou verzi algoritmu Blowfish. Je používána zejména pro šifrování hesel, v základu např. v operačních systémech OpenBSD nebo SUSE Linux, kdy OpenBSD je obecně používán právě díky své bezpečnosti. BCrypt využívá kryptografickou sůl, která je velmi účinná proti útokům metodou Rainbow Table, taktéž se jedná o adaptivní algoritmus, který je schopen zajistit stejnou úroveň zabezpečení i se zvyšujícím se výkonem počítačů a je tedy velmi odolný vůči útokům hrubou silou. JBCrypt je implemetace BCrypt algoritmu v jazyce Java. [20]

3.8 Adobe Photoshop

Photoshop je rastrový grafický editor, který jsem využil při tvorbě loga a pozadí pro spouštěcí aplikaci JavaFX. Jako logo (Obr. 3.2) jsem zvolil téma krychle, jelikož se celý herní svět skládá z krychlí a na pozadí se vyskytují hvězdy a různobarevné útvary, jaké bychom si v kosmu představili. [21]

3.9 Eclipse

Eclipse je open source vývojové prostředí pro jazyk Java. Výhodou je, že nabízí celou řadu open source pluginů, které zjednodušují práci. V mém případě jsem využil pluginy na podporu FXML a SceneBuilderu. Jednalo se o mé primární vývojové prostředí, které mi bylo doporučeno i mým vedoucím bakalářské práce. [22]



Obrázek 3.2: logo The Other Kosmos

3.10 DataGrip

DataGrip je systém správy databáze pro vývojáře, rovněž od firmy JetBrains. Je vytvořen pro tvorbu dotazů, tabulek a databází. Podporuje celou řadu databázových enginů a má příjemné uživatelské prostředí. Nevýhodou je, že není poskytován zdarma, výjimku tvoří studenti. Já jsem byl se softwarem obeznámen už z jiných databázových předmětů a vyhovovala mi jeho přehlednost a flexibilita. [23]

3.11 H2

H2 je relační databáze napsaná v Javě. Výhodou je, že může fungovat jako vestavěná přímo v aplikaci nebo serverově, což nám pro naše účely dává větší svobodu toho, co chceme s Other Kosmosem dělat, nabídnout singleplayer a v dalším vývoji i funkční multiplayer s jednou centrální databází.

Další výhoda je rychlost databáze, níže je umístěna tabulka s hodnotami, které porovnávají rychlost H2 databáze s jinými databázovými enginy. Je open source, tudíž celý projekt udržíme nezávislý na komerčně dostupném softwaru a není potřeba vynakládat zbytečné finance nebo se bát, že se cesta, kterou další vývoj povede diametrálně změní. Podporuje taktéž standartní SQL, JDBC API a je velice bezpečný. [24]

Tabulka obsahuje 4 testovací případy. Simple benchmark využívá jednu tabulku a na ní hodně jednoduchých aktualizací a mazání. BenchA je velmi podobný TPC-A testu a měří se jedno připojení s jedním vláknem. BenchB je podobný TPC-B testu, je navázáno několik připojení (jedno vlákno na připojení). BenchC je podobný TPC-C testu, ale je navázáno pouze jedno připojení s jedním vláknem. [25]

Tabulka 3.1: Rychlost vestavěných databází [25]

Pokus	jednotka	H2	HSQLDB	Derby
Simple: Inicializace	ms	1019	1907	8280
Simple: Dotaz (náhodný)	ms	1304	873	1912
Simple: Dotaz (sekvenční)	ms	835	1839	5415
Simple: Aktualizace (sekvenční)	ms	961	2333	21759
Simple: Smazání (sekvenční	ms	950	1922	32016
Simple: Využití paměti	MB	21	10	8
BenchA: Inicializace	ms	919	2133	7528
BenchA: Transakce	ms	1219	2297	8541
BenchA: Využití paměti	MB	12	15	7
BenchB: Inicializace	ms	905	1993	8049
BenchB: Transakce	ms	1091	583	1165
BenchB: Využití paměti	MB	17	11	8
BenchC: Inicializace	ms	2491	4003	8064
BenchC: Transakce	ms	1979	803	2840
BenchC: Využití paměti	MB	19	22	9
Spuštěno výpisů		1930995	1930995	1930995
Celkový čas	ms	13673	20686	105569
Výpisy za sekundu		141226	93347	18291

Tabulka 3.2: Rychlost databází komunikujících mezi klientem a serverem [25]

Pokus	jednotka	H2	HSQLDB	Derby	PostgreSQL	MySQL
Simple:Inicializace	ms	16338	17198	27860	30156	29409
Simple:Dotaz(náhodný)	ms	3399	2582	6190	3315	3342
Simple:Dotaz(sekvenční)	ms	21841	18699	42347	30774	32611
Simple:Aktualizace(sekvenční)	ms	6913	7745	28576	32698	11350
Simple:Smazání(sekvenční)	ms	8051	9751	42202	44480	16555
Simple:Využití paměti	MB	22	11	9	0	1
BenchA:Inicializace	ms	12996	14720	24722	26375	26060
BenchA:Transakce	ms	10134	10250	18452	21453	15877
BenchA:Využití paměti	MB	13	15	9	0	1
BenchB:Inicializace	ms	15264	16889	28546	31610	29747
BenchB:Transakce	ms	3017	3376	1842	2771	1433
BenchB:Využití paměti	MB	17	12	11	1	1
BenchC:Inicializace	ms	14020	10407	17655	19520	17532
BenchC:Transakce	ms	5076	3160	6411	6063	4530
BenchC:Využití paměti	MB	19	21	11	1	1
Spuštěno výpisů		1930995	1930995	1930995	1930995	1930995
Celkový čas	ms	117049	114777	244803	249215	188446
Výpisy za sekundu		16497	16823	7887	7748	10246

Kapitola 4

Rozšíření projektu Other Kosmos

4.1 O čem je Other Kosmos?

> „Vývoj aplikace ‚The Other Kosmos‘, jejímž dlouhodobým cílem je vytvořit herně-výukové simulační prostředí inspirované hrou Minecraft, které umožní absolutní přetváření simulovaného světa s hlavní podporou rozšiřitelnosti. Prostředí by mělo umožňovat vícevláknovou simulaci světa, řízené distribuování a aplikování rozšiřujících modulů. Výsledný simulátor by následně neměl sloužit jen k zábavě, ale i k výuce ať už na úrovni základních a středních škol, například ve fyzice (elektřina), tak i na úrovni vysokých škol, například rozmístění a monitorování chování celulární sítě, nebo počítačových bezdrátových sítí.“ [26]

4.2 Původní stav

Projekt Other Kosmos využíval relační databázový systém SQLite a objektově relační mapovací technologii j00Q, které v předchozím vývoji splňovaly, to co po nich bylo požadováno, ovšem pro další vývoj bylo potřeba tyto dva systémy nahradit něčím více flexibilním. Problém u j00Qu byl ten, že je poskytována pro něho podpora jen u těch nejnovějších verzí různých databázových systémů. Taktéž zde nebylo přítomné automatické generování tabulek, tudíž vše bylo potřeba dělat ručně a to by pro uživatele, ale i pro ostatní vývojaře, kteří se nutně nemusí vyznat v databázích byl problém. Z tohoto důvodu jsme zvolili EclipseLink jako alternativu s možností automaticky generovat tabulky.

SQLite splňoval podmínky pro lokální vývoj na vlastním PC, ovšem do budoucna by bylo stejně potřeba tento systém nahradit něčím jiným a proto jsme zvolili H2, který nabízí možnost lokální vestavěné databáze ale i serverového řešení.

V databázi byly uložené záznamy, které pracovaly s NPC a zde byl problém v tom, že se jednalo o diplomovou práci, která nebyla moc dobře zdokumentována z pohledu databáze. Poskytnuté záznamy z původní verze byly nic neříkající a mnohdy nekompletní, spuštění po krátkém čase vracelo

errory. Případně docházelo k zpomalení aplikace nebo až k jejímu ukončení. Nebylo možné jednoduše navazovat na započatou práci, taktéž celý systém starající se o chování NPC byl velice komplexní. Zde se počítalo se strojovým učením jako je např. učení chůze atd., tím pádem nešlo lehce předvídat jak se databáze bude za určitých podmínek chovat. To stížilo celkovou práci v pochopení podstaty jak věci fungují a jak byly zamýšleny, tedy být schopen je dále rozšiřovat.

Aplikace neukládala žádná uživatelské data, ani zde nebyla možnost jak s nima pracovat.

4.3 Nové funkcionality

4.3.1 Uživatelské účty

Ze všeho nejdřív jsem se začal věnovat implementaci uživatelských účtu a vlastních postav a jejich následné zařazení do simulovaného světa. Postavy jsou vždy propojeny s konkrétním uživatelským účtem a světem ve kterém se nachází. Každý svět má svou vlastní postavu, ke které se dostanete po přihlášení ke svému účtu a zvolení jednoho ze světů, ve kterém chcete momentálně hrát.

Bylo potřeba k uživatelům vymyslet nějaké atributy, prozatím každý účet má základní atributy jako jméno, heslo a případně jestli je administrátorem. Hesla jsou šifrované pomocí JBCryptu, využívající vyspělý šifrovací algoritmus BCrypt. Počítá se s možností dalšího rozšíření dle potřeby.

4.3.2 Postavy

Postavy v sobě uchovávají základní informace jako je poslední poloha před ukončením programu nebo věci v inventáři - jsou zde různé typy inventářů. Moje řešení poskytuje lehkou rozšiřitelnost při přidání dalších. Je zde prostor pro rozšíření uchovávaných informací, počítá se s ukládáním základních statistik jako je hlad, životy a zbroj, ovšem na této části pracuje v momentální době jiný vývojář.

4.3.3 Nehráčské postavy

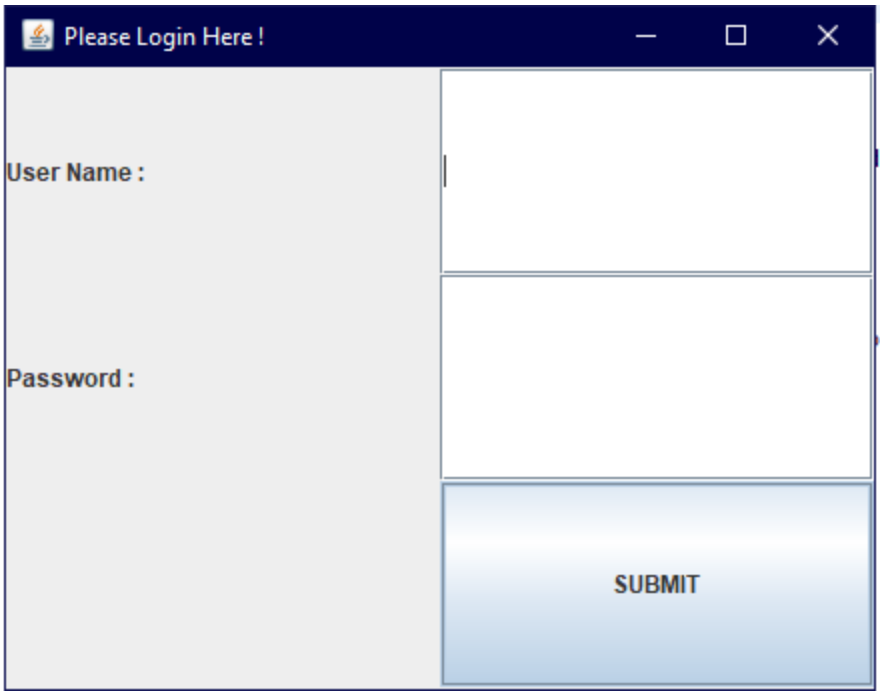
Pro správu nehráčských postav bylo vytvořeno prostředí v JavaFX. Zde se počítá s možností přidat do světa entitu. Entita je druh NPC, které může být zároveň i zadavatelem úkolu. Druhy jsou NPC o určité populaci (Obr. 4.1), které mají určité vlastnosti, např. jsou různě inteligentní, mají rozdílnou sílu atd. Jejich funkčnost jsem pozměnil, aby byly více praktické a využitelné ve hře. Nově lze nastavit pevné pozice pro entity i druhy a došlo k přidání nových modelů.

4.3.4 Spouštěcí aplikace

Bylo potřeba nejprve otestovat přihlášení, k tomu sloužila provizorní přihlašovací aplikace (Obr. 4.2) využívající zastaralou technologii SWING s velmi primitivním grafickým rozhraním. V této době ještě nebyla plně rozhodnuta finální podoba přihlašovací aplikace a smysl této aplikace byl pouze

testovací. Následně byla vytvořena stabilní verze (Obr. 4.3) pomocí JavaFX, která umožňuje vytvořit moderní, flexibilní spouštěč s dalšími možnostmi, jako je registrace uživatele přímo v aplikaci a pro administrátory jednoduchou správu NPC (Obr. 4.4, 4.5) .

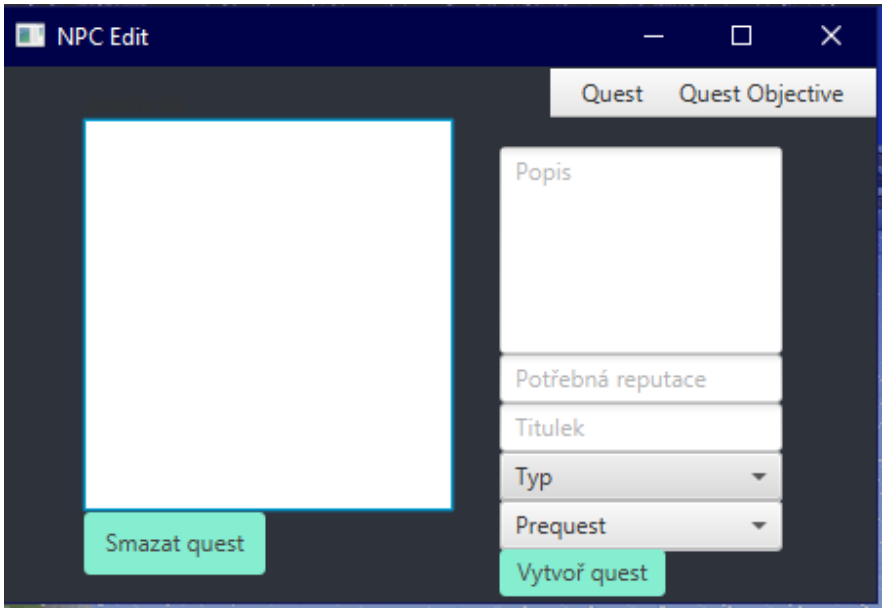
Obrázek 4.1: Různé populace



Obrázek 4.2: Testovací přihlašovací aplikace

Obrázek 4.3: Finální přihlašovací aplikace

Obrázek 4.4: Jedno z menu pro správu NPC



Obrázek 4.5: Menu pro správu úkolů

Kapitola 5

Implementace

5.1 Třídy CreatureDAO a CreatureModel

Tyto třídy byly vytvořené v rámci diplomové práce mého kolegy v minulém roce. Bylo potřeba určité věci přidat a doladit funkčnost. Byl přidán nový atribut model, který udává jaký model by měl být použit při vyobrazení dané entity v rámci herního světa.

Tato funkčnost v rámci entit nebyla uplně dokonalá, jelikož se zde nacházely pouze 2 modely a můj předchůdce namapoval v jiných třídách určité funkčnosti na tyto konkrétní modely, tudíž např. bylo pevně dané, že entita pouze s modelem lučištníka bude schopna nabízet ostatním postavám úkoly. Aplikace ovšem nefungovala dobře, pokud by měl daný úkol nabízet model rytíře.

Musel jsem tedy změnit celkový přístup. Třída CreatureType obsahovala tři výčty - **ARCHER**, **KNIGHT** a **PROCEDURAL GENERATED**, podle kterých se dané NPC chovají a mají přiřazený model. Namapované vlastnosti lučištníka a rytíře jsem ponechal a modely měním přímo při návratové metodě modelu podle záznamu v databázi. Bylo na místě pojmenovat dané výčty přilehavějšími jmény, aby nedošlo k mýlce. ARCHER byl přejmenován na **QUEST GIVER** a KNIGHT na **FIGHTER**.

Přidal jsem tři nové modely, které jsem našel na internetu a vyhovaly k mým testovacím účelům. LibGDX upřednostňuje použití modelů s formátem g3db, ovšem žádné takové jsem nemohl najít a když jsem se pokoušel o konverzi z formátu obj na g3db výsledek nebyl uspokojivý. Nakonec jsem tedy použil formát obj i když zde se projevují také určité nedostatky ve formě špatného mapování textur na modelech.

5.2 Třídy SpeciesDAO a SpeciesModel

Třídy, které slouží k ukládání a načítání automaticky vytvořených náhodných druhů, tento koncept byl rovněž implementován mým předchůdcem a nejednalo se o řešení, které dobře vyhovuje aktuál-

ním požadavkům, z toho důvodu, že druhy využívaly konceptu strojového učení. Samotné strojové učení je nutno spouštět samostatně a to celý proces komplikuje především při rozšiřování projektu.

Druhy se generovaly neustále při spuštěné aplikaci, rovněž k nim se generovaly části těl, které byly ukládána pomocí tříd BodyPartsDAO a BodyPartsModel. Proto, aby druhy v původní verzi fungovaly, bylo potřeba jim vytvořit tělo a naučit je chodit. K tomu sloužila aplikace WalkingGym, od které se upustilo, jelikož její použití nebylo moc praktické.

Od generování těl se rovněž odstoupilo a bylo potřeba jim náhodně přiřazovat modely, které jsou dostupné. Přidal jsem nový atribut model a pomocí náhodné funkce vybírám z pole modelů, která obsahuje názvy dostupných modelů. Dále jsem odstranil kód pro generování za běhu hry a náhodné generování jsem přizpůsobil k tomu, aby bylo možné generovat druhy v prostředí mé JavaFX aplikace. Aktivitní diagram na obrázku 5.1.

5.3 Třídy SpawnDAO a SpawnModel

Třídy jsou zodpovědné za ukládání informací o tom, kde se má konkrétní entita nebo druh v herním světě objevit. Tyto třídy byly také vytvořené mým předchůdcem, ale neměly žádné využití, jelikož zde chyběla logika, která by byla schopna s danýma informacema pracovat.

Bylo tedy potřeba implementovat pravidla při kterých se daná entita objeví na určitém místě v herním světě a k tomu bylo potřeba rozšířit o funkčnost třídu CreatureSpawner přidáním podmínek pro vytváření NPC.

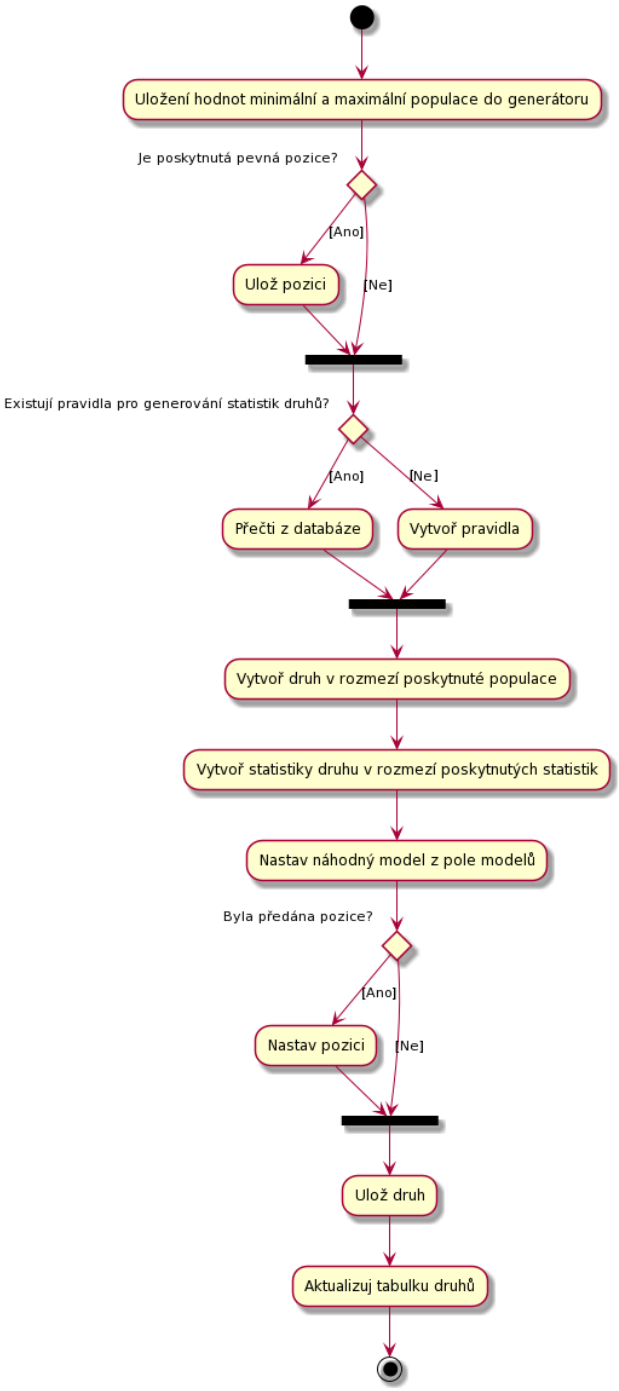
Jsou zde další atributy od mého předchůdce, které mají do budoucna sloužit jako pravidla chování pro určité entity nebo druhy, ovšem jednalo se o věci, které by v momentálním stavu neměly na chování daných nehráčských postav žádný vliv, tudíž se počítá s dalším rozšířením do budoucna.

5.4 Třídy QuestDAO a QuestModel

Třídy slouží k vytváření nových úkolů pro herní postavy. Zadavatelé úkolů můžou být pouze entity ze třídy CreatureModel. Každá entita může nabízet několik úkolů.

5.5 Třídy QuestObjectiveDAO a QuestObjectiveModel

K úkolům se dále vážou třídy, které slouží k uchovávání informací o jednotlivých dílčích úkolech pro splnění celkového úkolu, těch může mít jeden úkol několik.



Obrázek 5.1: Aktivitní diagram vytvoření druhů

5.6 Třídy StatSetModelDAO a StatSetModel

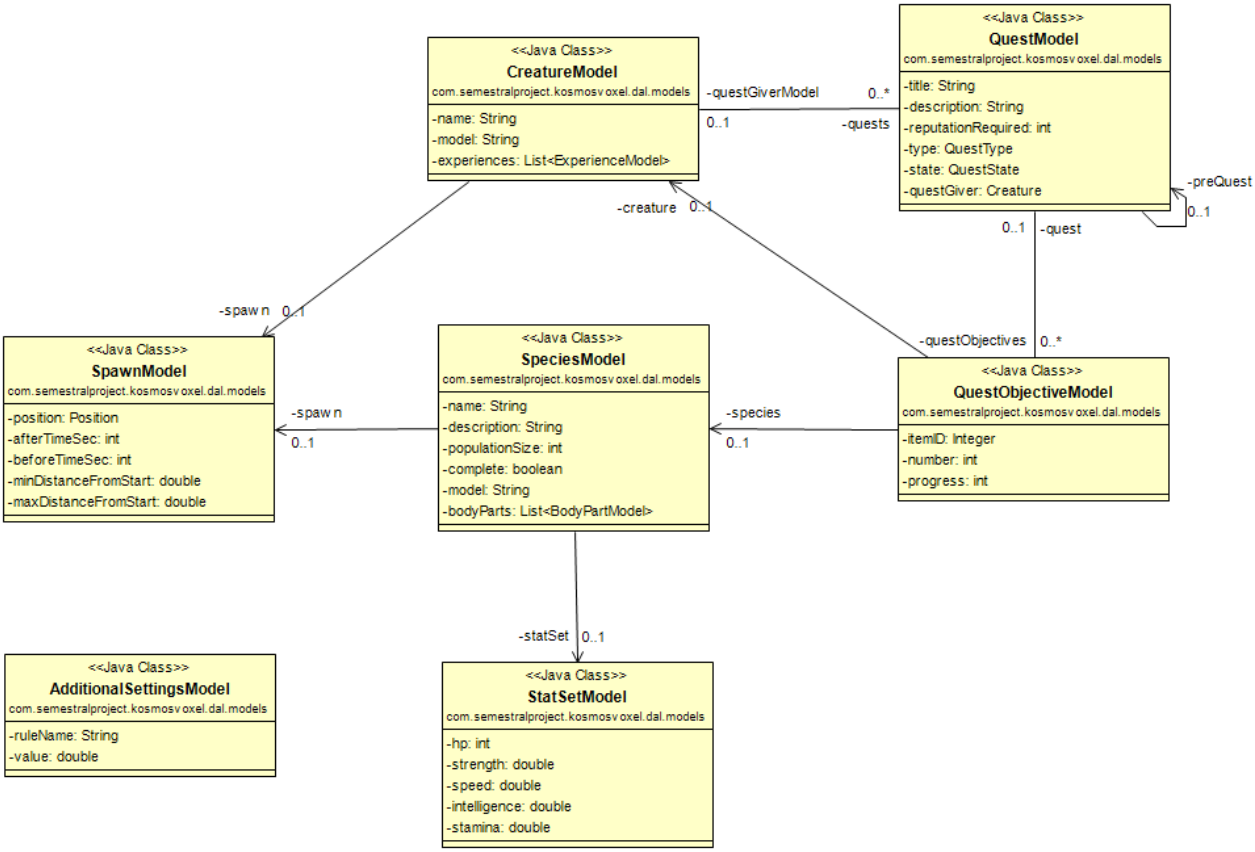
Slouží k uchovávání vygenerovaných statistik druhů. Statistiky jsou generovány ve třídě SpeciesGenerator, která byla upravena tak, aby hodnoty mezních proměnných sloužících při generování bylo možné změnit v databázi. Více v sekci 5.7

5.7 Třídy AdditionalSettingsDAO a AdditionalSettingsModel

Význam těchto tříd spočívá v možnosti přidávat pravidla, podle kterých se bude aplikace chovat, pravidla jsou ukládána v databázi.

Nyní obsahuje pravidla pro náhodné generování statistik od generovaných druhů. Předtím byly tyto pravidla pevně dané v aplikaci, nyní lze po změnění hodnot v databázi dosáhnout jiných hodnot v generování.

Do budoucna se počítá s rozšířením působnosti těchto tříd, poskytující kontrolu nad určitými atributy mimo kód.



Obrázek 5.2: Třídní diagram

5.8 Třídy UserDAO a UserModel

Třídy pracují s uživatelskými informacemi, jako jsou herní nick a heslo, obsahuje metody pro registraci nového uživatelského účtu a také přihlášení, je zde použit JBCrypt. Jednotlivé metody jsou volány ve třídě FXMLDocumentController.

```
if(getUserByNickname(nickname) == null) {
    UserModel user = new UserModel(nickname, password);
    save(user);
    return true;
}
return false;
```

Listing 5.1: Metoda pro registraci

```
UserModel user = getUserByNickname(nickname);
if(user != null) {
    if(BCrypt.checkpw(password, user.getPassword())) {
        return user;
   }
    else {
        throw new Exception("Neplatné údaje");
   }
}
throw new Exception("Uživatel neexistuje");
```

Listing 5.2: Metoda pro přihlášení

5.9 Třídy CharacterDAO a CharacterModel

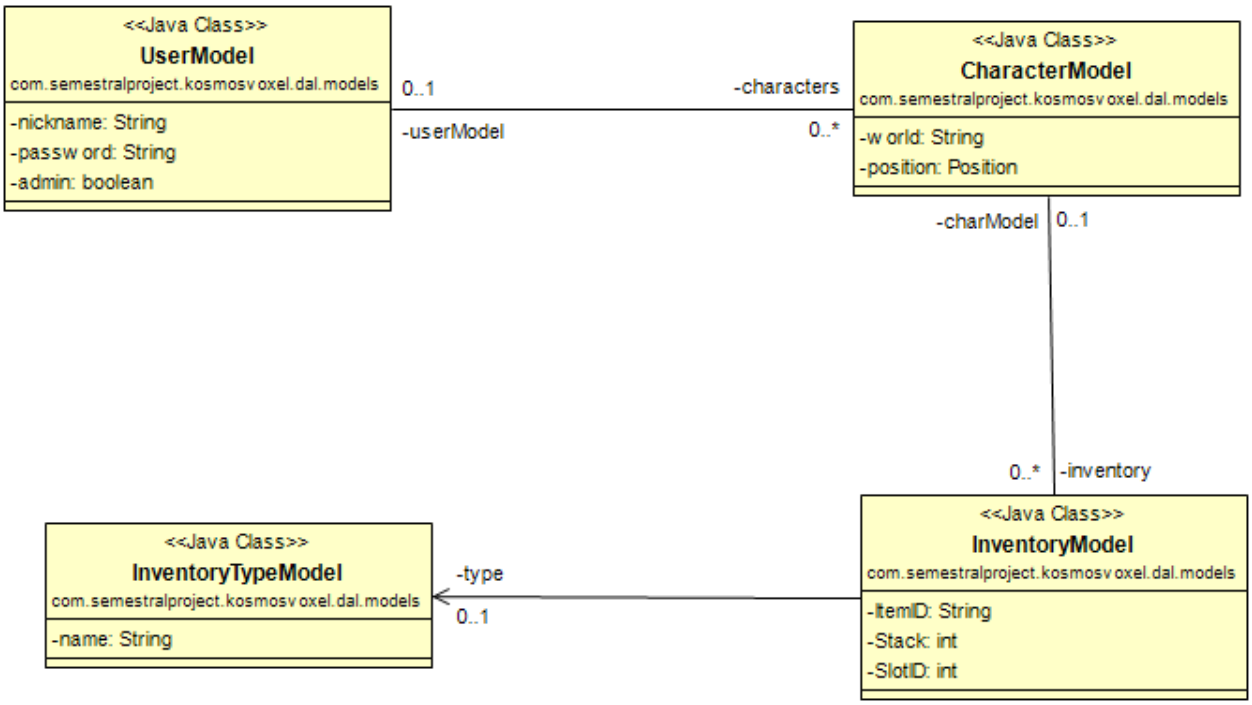
Pracují s informacemi o herních postavách, jako je zařazení do konkrétního světa a poslední pozice. Jsou navázané na inventář.

5.10 Třídy InventoryDAO a InventoryModel

Třídy pracují s uživatelským inventářem. Každá hráčská postava ve světě má záznamy v databázi reprezentující jednu položku v inventáři. Záznamů pro jednu postavu existuje tolik, kolik existuje míst ve všech inventářích, tedy každá postava má unikátní záznam pro danou pozici v inventáři. Tento záznam v případě, že neobsahuje žádnou položku je **null**.

5.11 Třídy InventoryTypeDAO a InventoryTypeModel

Třídy sloužíci pouze k identifikaci jednotlivých inventářů, do budoucna použitelné pro rozšíření o další inventáře.



Obrázek 5.3: Třídní diagram 2

5.12 Třída WorldSelectionScreen

Tato třída byla vytvořena kolegou taktéž pracujícím na tomto projektu a v jeho případě v rámci své bakalářské práce vytvořil tuto třídu, která funguje jako obrazovka pro správu světů. Zde je možné vytvořit nový svět nebo přejít už do existujícího, zde bylo mou prací vytvořit asociace mezi konkrétními účty a jejich postavami a světy.

Při vytvoření světa se vytvoří nový záznam v tabulce, který vytvoří postavu pro daný svět. Při zvolení už existujícího světa je potřeba kontrolovat, jestli záznam v tabulce už neexistuje, jelikož pokud vytvoříme svět na jiném účtu a přihlásíme se na druhém, tak světy už existují, tudíž je potřeba vytvořit novou asociaci pro jiný účet znovu.

```
if(DataSourceFactory.getInstance().getCharacterDAO().getChar(MainClass.user,
    worldName) == null)
{
DataSourceFactory.getInstance().getCharacterDAO().createNew();
```

```
}
else
{
MainClass.character = DataSourceFactory.getInstance().getCharacterDAO().getChar(
    MainClass.user, worldName);
}
```

Listing 5.3: Podmínka při které se přiřadí nebo vytváří nové postavy

5.13 Třída Voxel

Třída Voxel obecně slouží k nastavení herního světa, např. se zde nachází metody pro inicializaci kamery hráče nebo vykreslování modelů. Mým úkolem bylo tuto třídu rozšířit o načítání položek do jednotlivých inventářů z databáze.

5.14 Třída Player

Třída je zodpovědná za funkcionalitu hráčské postavy, nachází se zde metody měnící stav daného hráče, např. boj, skok, procházení herních menu, namapování kláves na určité funkce atd. Rozšířil jsem zde funkci tlačítka ESCAPE, které po zmáčknutí uloží aktuální stav postavy - poslední známou pozici a položky v inventáři do databáze.

5.15 GUILauncher a FXMLDocumentController

GUILauncher je soubor typu fxml, který uchovává informace o jednotlivých atributech a odkazuje na jejich funkce, které jsou dále rozvíjeny v třídě FXMLDocumentController.

Tyto soubory tvoří spouštěcí aplikaci, ve které je možné registrovat nový účet a přihlásit se. Pro administrátory nabízí i správcovské prostředí.

Před registrací zde probíhá jednoduchá kontrola vstupů. Po registraci je vytvořen záznam v tabulce UserModel. Při přihlášení opět probíhá kontrola vstupů a dochází k porovnávání hesel.

Pokud nejsme administrátoři tak se nic dalšího neděje a spustí se nám hra, ovšem pokud máme administrátorské oprávnění změní se v aplikaci okno a můžeme přepínat mezi tvorbou nových druhů (v aplikaci Species) nebo entit. Zde, pokud si ve hře vybereme svět můžeme získat i aktuální pozici naší postavy a v tom případě vytvořit dané NPC na konkrétním místě. Pro entitu je zde možnost vybrat i zobrazovaný model, pro druhy je zde možnost specifikovat minimální a maximální velikost populace jednoho druhu. Po vytvoření dané entity nebo druhu se nám zobrazí v příslušné tabulce a můžeme je i mazat nebo v případě entit rozšířit o další funkce, čehož se bude týkat další kapitola.

5.16 EditNPC a FXMLDocumentControllerEditNPC

EditNPC je soubor typu FXML a uchovává informace, se kterýma pracuje kontroler FXMLDocumentControllerEditNPC. Toto okno se vyvolá po zmáčknutí tlačítka "Přidat Quest", kde se zobrazí nabídka, která umožňuje vytvořit pro zvolenou entitu úkol, který bude nabízet ostatním postavám.

Zde je administrátor nucen vyplnit 4 povinné kolonky a jednu libovolnou a tou je„Prequest“, který slouží k přiřazení závislosti na předešlý úkol (pokud chceme vytvořit sérii na sobě závislých úkolů). Informace z těchto kolonek se pak následně ukládají jako hodnoty atributů do tabulky QuestModel. To ovšem ještě netvoří celkový úkol, ve hře se sice zobrazí, ovšem bez jasného cíle co má postava udělat.

Je proto nutné v menu zobrazit Quest Objective a vytvořit minimálně jeden pro tento úkol. Zde se opět nacházejí kolonky, kdy všechny nejsou povinné, záleží na kontextu úkolu. „QuestID“, „Progress“ a „Počet“ je nutné přiřadit vždy. „QuestID“ slouží k identifikaci toho, ke kterému úkolu daný „objective“ přiřazujeme. Podle toho jaký typ úkolu jsme vybrali při tvorbě úkolu vybereme, jestli má postava sbírat určitý block (ItemID) nebo jestli má např. vyvraždit určitou entitu (CreatureID) nebo druh (SpeciesID). Informace z těchto kolonek se pak následně ukládají jako hodnoty atributů do tabulky QuestObjectiveModel.

5.17 Třída Filters

Třída filters obsahuje konkrétní implementované objekty TextFormatter. Ty slouží především k filtrování obsahu, který musí být číselný, tedy jejich funkce je taková, že v rámci objektu TextField poskytující knihovnami JavaFX lze nastavit pro tento objekt textový filtr.

V mém případě bylo potřeba vytvořit 2 textové filtry, jeden pouze s přirozenými čísly, využití např. v textovém poli „reputace“. Druhý filtr byl vytvořen pro kontrolu čísel s plovoucí desetinnou čárkou, při vyplňování tří textových polí, které slouží k přiřazení pevné pozice dané entity, pro tento účel jsem využil regulární výrazy, jelikož bylo potřeba povolit specifické znaky jako „-“ a „.“.

Tato třída vznikla hlavně proto, aby nebylo potřeba v kódu vytvářet znova daná pravidla, ale pouze vytvořit novou instanci a přiřadit k danému textovému poli.

```
public TextFormatter<Object> floatTextFormatter = new TextFormatter<>(change -> {
    Pattern pattern = Pattern.compile("^-?[0-9.]*$");
    return pattern.matcher(change.getControlNewText()).matches() ? change : null;
});
```

Listing 5.4: Objekt kontrolující číslo s plovoucí desetinnou čárkou

```
public TextFormatter<Object> integerTextFormatter = new TextFormatter<>(c -> {
if (c.isContentChange()) {
```

```
    if (c.getControlNewText().length() == 0) {
        return c;
        }
    try {
        Integer.parseInt(c.getControlNewText());
        return c;
    }catch (NumberFormatException e) {
    }
    return null;
}
return c;
});
```

Listing 5.5: Objekt kontrolující přirozené číslo

5.18 Další práce

Obecně muselo být pozměněno mnoho dalších tříd, ovšem více pozornosti jsem věnoval jen těm, kde bylo potřeba udělat větší změny.

V rámci druhů bylo potřeba přepsat funkčnost několika tříd, aby nedocházelo ke generování těla pomocí strojového učení. Dále bylo potřeba opravit chyby, které nastaly při nekorektním spojení mnoha různých částí projektu. Vytvořil jsem několik JPQL dotazů, pro ulehčení práce. Bylo potřeba mnoho věcí otestovat a pochopit jak vlastně byly zamýšleny, případně rekonstruovat funkčnost, hlavně vše co se týkalo NPC a bylo vytvořeno předchozím vývojářem.

Kapitola 6

Závěr

Práce na projektu mě bavila a zároveň mi dala trochu náhled do toho, jak funguje práce ve více lidech na společném projektu. Využil jsem zde hlavně vědomosti z předmětů databázových systémů a vývoje informačních systémů. Obecně mě práce s databázemi v praxi baví a chtěl bych se v tom i nadále zlepšovat.

Za přínos považuji i to, že jsem si vyzkoušel, jak se tvoří komplexnější hra a že je potřeba se naučit číst cizí kódy a pochopit jejich propojení v rámci celého projektu. Do budoucna plánuji pokračovat na vývoji. Přidat určité ochranné prvky co se týče registrace, aby nedocházelo např. k zahlcení databáze v rámci útoku hrubou silou, namapovat NPC na konkrétní světy atd.

Literatura

1. ČERNÍN, Karel; DANĚK, Tomáš; SOVADINOVÁ, Marcela. *Teorie RPG aneb Na hranicích fantazie*. [N.d.]. Dostupné také z: `http://www.taria.unas.cz/files/TeorieRPG.pdf`.
2. *Co je to Dračí Doupě?* [Online]. Altar, 2002 [cit. 2021-04-11]. Dostupné z: `http://www.dracidoupe.cz/index.php?rub=cojetodrd%5C&skin=dark`.
3. *Sada kostek pro RPG Classic* [online] [cit. 2021-04-28]. Dostupné z: `https://www.fantasyobchod.cz/sada-jednoduchych-elfskych-kostek`.
4. *Non-Player Character* [online] [cit. 2021-04-11]. Dostupné z: `https://chrono.fandom.com/wiki/Non-Player_Character`.
5. *Nim* [online] [cit. 2021-04-11]. Dostupné z: `https://www.algoritmy.net/article/30057/Nim`.
6. BOURG, M. David; SEEMANN, Glenn. *AI for game developers*. Vyd. 1. Cambridge: O´Reilly, 2004. ISBN 0-596-00555-5.
7. JAKOB, Michal. *Umělá inteligence v počítačových hrách* [online] [cit. 2021-04-11]. Dostupné z: `https://www.scienceworld.cz/technologie/umela-inteligence-v-pocitacovych-hrach-1-2333/`.
8. *Minecraft* [online] [cit. 2021-04-28]. Dostupné z: `https://www.minecraft.net/`.
9. DARWIN, Ian F. *Java: kuchařka programátora : [vzory a řešení pro vaše aplikace]*. Vyd. 1. Brno: Computer Press, 2006. ISBN 80-251-0944-5.
10. STÄRK, Robert F.; SCHMID, Joachim; BÖRGER, E. *Java and the Java virtual machine: definition, verification, validation*. Berlin: Springer, 2001. ISBN 3540420886.
11. LINDHOLM, Tim; YELLIN, Frank; BRACHA, Gilad; BUCKLEY, Alex; SMITH, Daniel. *The Java® Virtual Machine Specification* [online]. 2021 [cit. 2021-04-11]. Dostupné z: `https://docs.oracle.com/javase/specs/jvms/se16/html/index.html`.
12. *JVM Garbage Collectors* [online] [cit. 2021-04-11]. Dostupné z: `https://www.baeldung.com/jvm-garbage-collectors`.

13. ALTVATER, Alexandra. *What is Java Garbage Collection? How It Works, Best Practices, Tutorials, and More* [online]. 2017 [cit. 2021-04-11]. Dostupné z: https://stackify.com/what-is-java-garbage-collection/.

14. *Java$_v$irtual$_m$achine$_a$rchitecture.svg* [online] [cit. 2021-04-28]. Dostupné z: https://pcchip.hr/softver/korisni/virtualizacija/attachment/java_virtual_machine_architecture-svg-3/.

15. FUSEK, Zdeněk. *Grafické uživatelské prostředí v JavaFX* [online]. Brno, 2013 [cit. 2021-04-21]. Dostupné z: http://hdl.handle.net/11012/28191. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací.

16. *Features* [online] [cit. 2021-04-21]. Dostupné z: https://libgdx.com/features/.

17. *JavaFX Scene Builder: A Visual Layout Tool for JavaFX Applications* [online] [cit. 2021-04-21]. Dostupné z: https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html.

18. *Úvod - Základy systému Git* [online] [cit. 2021-04-11]. Dostupné z: https://git-scm.com/book/cs/v2/%C3%9Avod-Z%C3%A1klady-syst%C3%A9mu-Git.

19. *EclipseLink: Comprehensive open-source Java persistence solution addressing relational, XML, and database web services.* [Online] [cit. 2021-04-21]. Dostupné z: https://www.eclipse.org/eclipselink/#about.

20. PROVOS, Niels; MAZIÈRES, David. *A Future-Adaptable Password Scheme*. Monterey, California, USA, 1999. Dostupné také z: https://www.usenix.org/legacy/events/usenix99/provos/provos_html/.

21. *Adobe* [online] [cit. 2021-04-28]. Dostupné z: https://www.adobe.com/cz/products/photoshop.html.

22. *Desktop IDEs* [online] [cit. 2021-04-21]. Dostupné z: https://www.eclipse.org/ide/.

23. *Introduction* [online] [cit. 2021-04-11]. Dostupné z: https://www.jetbrains.com/help/datagrip/meet-the-product.html.

24. *Features* [online] [cit. 2021-04-11]. Dostupné z: https://www.h2database.com/html/features.html.

25. *Performance* [online] [cit. 2021-04-11]. Dostupné z: https://www.h2database.com/html/performance.html.

26. *The Other Kosmos* [online] [cit. 2021-04-11]. Dostupné z: https://swi.cs.vsb.cz/jezek/research-and-development/development/the-other-kosmos.html.